

SECURE MULTIPARTY COMPUTATION FOR GEOMETRIC PROBLEMS

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE BACHELOR OF SCIENCE IN COMPUTATIONAL
MATHEMATICS

BY

MUKESH TIWARI

4th Year/ 2nd Semester



DEPARTMENT OF MATHEMATICS
SCHOOL OF SCIENCE
KATHMANDU UNIVERSITY
DHULIKHEL, NEPAL

JULY 2024

Dedication

To

My parents

Kiran Tiwari

&

Rameshwar Tiwari

and my sister

Manisha Tiwari

Declaration

I, Mukesh Tiwari, hereby declare that the work contained herein is entirely my own, except where stated otherwise by reference or acknowledgment, and has not been published or submitted elsewhere, in whole or in part, for the requirement for any other degree or professional qualification. Any literature, data or works done by others and cited within this project work has been given due acknowledgment and listed in the reference section.

Mukesh Tiwari

KU Registration number: 026615-19

Date:

Certificate

This project entitled “**Secure Multiparty Computation for Geometric Problems**” is carried out under my supervision for the specified entire period satisfactorily, and is hereby certified as a work done by the student, Mukesh Tiwari in partial fulfillment of the requirements for the degree of B.Sc. in Computational Mathematics, Department of Mathematics, Kathmandu University, Dhulikhel, Nepal.

Dr. Gokul K.C.

Associate Professor

Department of Mathematics

School of Science, Kathmandu University

Dhulikhel, Kavre, Nepal

Date:

APPROVED BY:

I hereby declare that the candidate qualifies to submit this report of the Mathematics Project (MATH 499) to the Department of Mathematics.

Prof. Dr. Rabindra Kayastha

Head Of Department

Department of Mathematics

School of Science

Kathmandu University

Date:

Acknowledgment

This work has been possible with the support and guidance of many. First and foremost, I would like to thank my supervisor, Dr. Gokul KC, for his excellent mentorship and supervision. I am thankful to the entire Department of Mathematics and the School of Science for constant guidance and motivation.

I was introduced to the field of Secure Computation from the lectures of Dr Vishal Saraswat and Mr Rohitkumar R. Upadhyay during the CIMPA school on 'Post Quantum Cryptography'. I owe my heartfelt gratitude to them and the entire team of CIMPA for the opportunity to explore this exciting area of research.

I am thankful to my dear friends Priyanka Panta and Ayam Basyal for their meticulous proofreading and always being there to listen to my ideas and provide feedback. This work has been possible due to the constant support and encouragement from my friends and family. I am thankful to you all.

I have relied on many open-source libraries and tools to compile information, generate figures and write code. I am grateful to the developers and contributors for their awesome

Thank you.

Mukesh Tiwari

Abstract

Secure multiparty computation evaluates functions using inputs from multiple parties such that inputs are kept private from others. Theoretical results have established the feasibility of secure evaluation of any computable function. Many problems have been implemented securely for practical applications. The point inclusion problem is to determine whether a given point lies inside the given polygon. In this paper, we use MPyC, a Python framework for secure multiparty computation that utilizes Shamir's secret-sharing scheme, to implement a secure point inclusion problem with the help of a trusted helper. This work helps parties securely evaluate point inclusion which can form the basis of many interesting applications.

Keywords: secure computation, Shamir's secret sharing,
secure computational geometry, privacy, multiparty computation

Outline of the Report

This Project work is divided into four chapters. Chapter-wise cameo description of the work is as follows

Chapter 1 provides introduction to secure computation as well literature review in context of secure computational geometric problems. It also discusses some existing frameworks for multiparty secure computation.

Chapter 2 includes preliminaries and description of secure point inclusion problem with worked out examples.

Chapter 3 provides description of program for secure multiparty point inclusion problem with its limitations and demonstration.

Chapter 4 discusses about future work and recommendations.

List of Figures

2.1	Example of secure addition using Shamir's secret sharing	13
2.2	Graph with Polygon and Points (generated with GeoGebra [®]) . .	16
3.1	Flowchart of Implementation	19
3.2	Example point.csv File	20
3.3	Example polygon.csv File	21
3.4	Execution for Trusted Helper	22
3.5	Execution by party containing point	22
3.6	Execution by party containing polygon	23

List of Tables

2.1	Tabulated values for the workout example of line point inclusion .	17
3.1	Roles and Inputs in Secure Point Inclusion	19

Contents

Dedication	iii
Declaration	iv
Certification	v
Acknowledgements	vi
Abstract	vii
List of Figures	ix
List of Tables	x
List of Abbreviations	xiii
1 Introduction	1
1.1 Background	2
1.2 Research Objectives	2
1.3 Literature Review	2
1.3.1 Secure Computation	2
1.3.2 Secure Geometric Computation	3
1.3.3 Open Source Libraries and Frameworks for MPC	3
1.3.4 Fairplay	4
1.3.5 FairplayMP	4
1.3.6 VIFF	4
1.3.7 MPyC	5
2 Preliminaries	6
2.1 Security of Multiparty Computation	6

2.1.1	Ideal/Real Paradigm	6
2.2	Trusted Parties	7
2.2.1	Trusted Third Party	7
2.2.2	Trusted Initializer	7
2.2.3	Trusted Helper	8
2.3	Adversary	8
2.3.1	Adversarial Behavior	8
2.3.2	Corruption Strategies	9
2.4	Cautionary Notes in use of MPC	9
2.5	MPC Primitives	10
2.5.1	Yao's millionaire Protocol	10
2.5.2	Oblivious Transfer	10
2.5.3	Secure Secret Sharing	11
2.5.4	Secure Addition using Shamir's Secret Scheme	12
2.5.5	Secure Point Inclusion Problem	13
3	Computer Implementation	18
3.1	Description of MPyC	18
3.2	Outline of Program	18
3.3	Demonstration	20
3.3.1	Input : Point	20
3.3.2	Input : Polygon	20
3.3.3	Running the code	21
3.4	Limitations	23
4	Future Work and Recommendation	24
4.1	Future Work	24
4.2	Conclusion	24
	Appendices	25
A	Python Implementation of Secure Point Inclusion Problem in MPyC	25

List of Abbreviations

MPC	Multi Party Secure Computation
SFDL	Secure Function Definition Language
SFE	Secure Function Evaluation
SHDL	Secure Hardware Definition Language
VIFF	Virtual Ideal Functionality Framework

Chapter 1

Introduction

In the age of information we live in, parties and agencies often want to collaborate to get results based on combined data but have constraints and reasons to maintain the privacy of their data as well. These goals may seem conflicting goals. However, we can have secure computation which maintains the privacy of the inputs while computing the desired results.

Consider a sealed-bid auction where multiple parties present their bid simultaneously for the product they are interested in. The goal is to identify the highest bidder. The bidders do not want to disclose their bid amount to anyone. In these scenarios, the historical solution has been to have a 'Trusted Third Party' who can reveal the highest bidder without disclosing the actual value of the bids. This solution is unsatisfactory as the 'trusted' third party may not be trustworthy and corruptible.

Abstractly, we need a method in which any number of parties collectively compute an agreed-upon function of their inputs with the condition that no party learns anything about the input of other fellow participants. These methods must ensure that all the inputs stay private. In literature, these protocols for these problems are termed 'Multiparty Secure Computation'. [1]

We shall explore this notion formally and see if it is mathematical feasibility. We shall also apply secure protocols to evaluate the point inclusion problem in a high-level programming language, Python.

1.1 Background

Cryptography is generally associated with just hiding messages. Donald Davis mentions that cryptographic techniques have been used to send diplomatic notes, war correspondence and love letters safe from snooping eyes. [2] The prevalence of cryptography as a practical tool throughout human civilization is indisputable.

In recent times, cryptography encompasses a much larger domain in which secure communication and secure computation play equally important roles. Secure communications and many other cryptographic problems are special cases of secure communication. [3]

1.2 Research Objectives

The objectives are as follows:

- (i) Study of protocols on secure multiparty geometric problems.
- (ii) Python implementation of protocol for secure point inclusion problem.

1.3 Literature Review

The history of secure computation is fairly recent yet huge theoretical breakthroughs have been made. Hence, the field of multiparty secure computing is rich in both theory and applications.

1.3.1 Secure Computation

In the 1970s, there was an interest in ideas like 'Mental Poker'. [4]. The same guys who gave us RSA (Adi Shamir, Ronald L. Rivest and Leonard M. Adelman) published an article describing the problem of two parties playing poker with only messages and no physical cards. They gave a protocol for achieving this feat. This was one of the initial uses of secure computation for a specific problem.

In 1982, Yao presented the classical paper [5] in which he introduced the general notion of secure computing and proposed the solution to the two-party Millionaire's Problem. A major leap was achieved by Goldreich when he showed that all computations can be done as Multiparty Secure Computation with an

honest majority. [6]. This was a great theoretical result but provided no practical and efficient implementation.

Since then, much work has been done in efficient implementation in various areas like secure data mining, secure auction and secure machine learning. There have been real-world applications employing secure computation. [7]. One prominent example was MPC-based data analysis to evaluate the wage gap in Boston. As described by Lapets et al. in [8], Corporations were hesitant to provide raw data due to privacy concerns. The MPC-based solution gave only aggregates as outputs while all other inputs remained private (even from the servers doing the computations).

1.3.2 Secure Geometric Computation

In the year 2001, Atallah et al. [9] explored the following problems in plain geometry from a secure multiparty point of view.

- Point Inclusion in a polygon
- Intersection of two polygons
- Closest pair amongst two sets of points
- Convex Hull from combined set of points from two sets

They considered the known protocols like Yao's millionaire protocol and devised some new geometric protocols that act as primitives in building securely computing the problems stated above.

This work was build upon by many researchers who extended it to problems like intersection of lines [10], intersection of circles and distance between points. [11][12]

The domain of geometric problems was extended to problems in solid geometry (i.e, in three dimensions) by Shundong Li et al. in 2014. He demonstrated a protocol for calculating the volume of a tetrahedron. [13]

1.3.3 Open Source Libraries and Frameworks for MPC

Secure Computation is not just an theoretical curiosity. It is very much applicable and already in use. Hastings et al. [14] have compiled a github repository contain-

ing the dockerized versions of many MPC frameworks along with examples and additional details which is available at <https://github.com/MPC-SoK/frameworks>.

We shall describe a few important frameworks that paved the path for MPyC, the framework used to implement secure point inclusion in this study.

1.3.4 Fairplay

As mentioned by Marcella Hastings et al., [14], Fairplay was the first publicly available MPC compiler. It was launched in 2014. Fairplay was introduced in the paper by Dahlia Malkhi et al. [15] in which they claimed it to be a generic secure function evaluation (SFE). It comprised of Secure Function Definition Language (SFDL) which is a high level language heavily inspired by C and Pascal that is used to write secure computation programs. Fairplay compiles the SFDL program into a one-pass Boolean Circuit that is represented using Secure Hardware Definition Language (SHDL). One of its limitations is the fact that it only supports two party secure computation.

1.3.5 FairplayMP

FairplayMP was introduced in the 2008 by the team of Assaf Ben-David, Noam Nisan and Benny Pinkas. The last two authors were also on the team behind Fairplay. In the paper introducing FairplayMP, they describe it as an extension to the Fairplay software which supports secure computation between two parties.

1.3.6 VIFF

Virtual Ideal Functionality Framework (VIFF) by introduced by Martin Geisler as his PhD work in 2007. VIFF 1.0 was released on Dec 2009 and contained many core functionalities that enabled one to write multiparty secure protocols in Python. [16]. Geisler notes the following three main features of VIFF: [17]

1. Asynchronous execution
2. Automatic parallel scheduling
3. Easy composability

The major advantage of VIFF was easy composability, which allowed for MPC protocols to be written without having to implement the primitives again and again.

1.3.7 MPyC

MPyC is a open-source python framework for secure multiparty computation based on the VIFF. It is based on Shamir’s secret sharing scheme as described in 2.5.3 and thus supports a security evaluation as long as the adversaries are passive and there exists an honest majority. It works asynchronously and hence can be easily used for implementing and testing protocols over the internet. [16].

The homepage of MPyc is hosted at <https://www.win.tue.nl/~berry/mpyc/>. The source code and the installation instructions can be found at <https://github.com/lschoe/mpyc>. The package is very well documented. The documents can be found at <https://lschoe.github.io/mpyc/> and <https://mpyc.readthedocs.io/en/latest/>.

Chapter 2

Preliminaries

2.1 Security of Multiparty Computation

Although an informal definition of secure computation has been provided, a formal and complete definition was provided by [18] in which they define secure computation by mimicking ideal evaluation, blending privacy and correctness, and requiring adversarial awareness. Their definition can be summarized as below:

2.1.1 Ideal/Real Paradigm

We consider two scenarios. In the first one, there is an external trusted and incorruptible third party 2.2.1 whom all parties trust. In this ideal world, the secure computation is achieved by sending all the inputs to the trusted third party privately. The parties then receive the output from this third party.

On the other hand, we have the real world where there is no trusted third party and all the computations are done by some protocol in which they all participate. We may assume that some are colluding or not honest as per our need. Then a protocol is said to be secure if it can emulate the ideal world.

This means that any attack to gain more information than their input and output is successful in the real protocol if and only if the ideal version of the same computation can also be compromised by the same attack.

More details of this ideal/real paradigm can be found at [19]

Although contained in the above definition, Lindell [20] explicitly describes some properties (not exhaustive) that a secure computation protocol must achieve:

1. **Privacy**

The only additional information that a party should gain from participating in the protocol should be the prescribed output.

2. **Correctness**

The output of the protocol is the evaluation of the predefined function.

3. **Independence of Inputs**

Each party must choose their input independently of the other parties.

4. **Guaranteed output delivery**

No party can prevent any other party to receive their output.

5. **Fairness**

Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs.

2.2 Trusted Parties

2.2.1 Trusted Third Party

Before Yao demonstrated the possibility of Secure Multiparty Computation without any trusted third party, the privacy of input was achieved by using a trusted third party. A trusted third party is an entity that provides no input in the computation but assists the other party in achieving their goal.

A traditional third party may simply collect all the inputs, perform the computation and then provide only the outputs to the parties. If the third party is honest and incorruptible, the computation achieves the goal of MPC. However, the practicality of finding such an ideal party and the fact that the trusted party also learns of the output makes this not desirable.

A capital firm handling IPO allocation of a company acts as a trusted party between the company and potential shareholders (applicants).

2.2.2 Trusted Initializer

Rivest [21] coined a new term '**trusted initializer**' to describe parties that participate only in a setup phase before the other parties may even have their inputs.

They may even go offline after the initial setup and thus do not learn about anything about the inputs or the outputs.

2.2.3 Trusted Helper

A third type of third party may be defined where they are online and actively participate in the protocol for the entire duration. However, they don't learn the inputs or the outputs in the process. They simply aid in the protocol.

However, it is important that they do not get corrupted. If enough fraction of the trusted helpers are corrupted by the Adversary, the protocol may no longer be secure.

In chapter 3, We shall utilize a trusted helper to implement secure version of point inclusion problem in python using MPyC protocol.

2.3 Adversary

In multiparty secure computation, we cannot practically assume that all parties will be honest and not try to collude or disrupt the process in some way.

2.3.1 Adversarial Behavior

Lindell [20] describes these three kinds of adversary based on what kind of actions they are allowed to take.

- **Semi-honest Adversaries**

These adversaries follow the protocol honestly but the corrupted parties do collude and share their respective states and thus all messages they have sent and received in order to gain more information than what they should know from the protocol. These are also known as '*honest but curious adversaries*' and *passive adversaries*.

- **Malicious Adversaries**

Malicious adversaries or '*Active adversaries*' can cause any corrupt parties to arbitrarily deviate from their specification as described in protocol. Security against such attacks means that no adversarial attack can succeed.

- **Covert Adversaries**

In this model, the corrupt parties can deviate from the protocol just like malicious adversaries. The difference lies that there is a specified probability that they will be caught. However, if they are not caught, they may be able to compromise other's inputs. These model can be useful in real life where a punishment or deterrence for those who deviate from protocol in an dishonest way can be enforced.

2.3.2 Corruption Strategies

Based on when and how the parties can be corrupted, Lindell [20] has classified adversary's strategy into three categories.

- **Static Corruption Model**

In this model, the honest and the corrupted parties are defined before hand and they don't change throughout the protocol execution.

- **Adaptive Corruption Model**

In adaptive corruption model, the adversary can choose who and when to corrupt based on execution at any time. However, any party once corrupted remains so throughout.

- **Proactive Corruption Model**

This is similar to adaptive model with the addition that a corrupted party may again become honest. Work by Canetti and Herzberg [22] explore this model of corruption strategy.

2.4 Cautionary Notes in use of MPC

Yehuda Lindell describes these two properties in his work [20], which are of paramount importance whenever one is dealing with any sort of secure computation.

- **Any inputs are allowed**

Although MPC can secure the input of one party from all other parties. It cannot be restrict the parties from inputting values that make no sense. for example, one can enter their salary as 10 trillion which is absurd.

- **MPC secures the process, but not the output**

The output of certain computation may also leak information about the inputs used during the computation. For example, computing sum from inputs two parties cannot be secured as one can deduct their value from the computed sum to get the other's value.

2.5 MPC Primitives

2.5.1 Yao's millionaire Protocol

Andrew C. Yao in his work that started the field of multiparty secure computation. [5], proposes a solution to the problem he posed. How can two millionaires determine who is richer without disclosing how much each party is worth?

This can be seen as a specific case of the general problem of secure computation with m parties where they all wish to compute $f(x_1, x_2, \dots, x_m)$ where x_i is the private input of the i^{th} party. In our case $m = 2$ and the function f is defined as

$$f(x_1, x_2) = \begin{cases} 0 & x_1 < x_2 \\ 1 & otherwise \end{cases}$$

2.5.2 Oblivious Transfer

Oblivious transfer is a protocol that forms the primitive of MPC. In 1981, Michael O. Rabin published a paper titled 'How to Exchange Secrets with Oblivious Transfer' [23].

This made it possible for Bob to receive from Alice either x_0 or x_1 without Alice having any idea which was the element that Bob desired. The value of other variable of Alice should however be private and Bob should have no way of knowing it.

This was expanded to the case where Alice has the set $\{x_1, x_2, \dots, x_n\}$ and Bob can inquire about any one entry x_i where $1 \leq i \leq n$ without disclosing the index i . [24]

2.5.3 Secure Secret Sharing

Secret sharing is an important primitive for many MPC protocols. In the implementation described in chapter 3, we shall use MPyC which utilizes this scheme to ensure that inputs are hidden from other parties but yet computation using those inputs is possible.

In 1979, Shamir [25] described a scheme to divide data into n pieces such that the data is easily reconstructed from k pieces of the data but no information is available from $k - 1$ pieces. Any such scheme is known as a (k, n) *threshold scheme*.

Definition 2.5.1 ((k, n) **threshold scheme**) [25]

A (k, n) *threshold scheme* is defined as a scheme that divides data D into n pieces D_1, D_2, \dots, D_n such that:

- (1) Knowledge of any k or more D_i pieces makes D easily computable.
- (2) Knowledge of any $k - 1$ or fewer D_i pieces leaves D completely undetermined (in the sense that all its possible values are equally likely)

In the same paper,[25] Shamir describes a (k, n) *threshold scheme* based on polynomial interpolation. Consider a random polynomial $q(x)$ of degree $k - 1$

$$q(x) = a_0 + a_1x + \dots + a_{k-1}x^{k-1}$$

and set $a_0 = D$ where D is our data.

We then define

$$D_i = q(i) \quad \text{for } 1 \leq i \leq n$$

.

Now we shall show this scheme satisfies the conditions laid out in section 2.5.1 by using the following well known lemma.

Lemma 2.5.1 $n + 1$ points (x_i, y_i) with unique x_i 's uniquely determine a polynomial of degree n . [26]

We can now share any k or more values of D_i and the recipient could easily generate the coefficients of the polynomial $q(x)$ by methods such as Lagrange interpolating polynomials [27] and evaluate it at 0 to get the value of D as

$$q(0) = a_0 = D$$

However sharing any less than k points will lead to infinitely many polynomials of degree k containing the given points. Thus, D is not revealed.

Hasting et al. [14] claims that most MPC rely on secret sharing schemes that can offer linearity. This property means that the sum of two secret shares is equal to the share of the sum.

2.5.4 Secure Addition using Shamir's Secret Scheme

Shamir's secret scheme not only allows us to share a secret but also allows us to achieve secure evaluation of values. We shall discuss secure addition with a worked out example.

Let two parties have value x and y respectively. To share the inputs securely, the parties construct two polynomials randomly.

Party 1 constructs:

$$f(x) = a_0 + a_1x \text{ where } a_0 = x \quad (2.1)$$

and then constructs the two shares as

$$S_x^1 = f(1) \quad (2.2)$$

$$S_x^2 = f(2) \quad (2.3)$$

Party 2 constructs:

$$g(x) = b_0 + b_1x \text{ where } b_0 = y \quad (2.4)$$

and then constructs the two shares as

$$S_y^1 = g(1) \quad (2.5)$$

$$S_y^2 = g(2) \quad (2.6)$$

After exchange of shares, Party 1 has $\{S_x^1, S_y^1\}$ which is $\{f(1), g(1)\}$ while party 2 has $\{S_x^2, S_y^2\}$ or $\{f(2), g(2)\}$. With the value of these secrets, there is no way for either party to find out the value of other party's input.

However, we can see that

$$x + y = a_0 + b_0 = f(0) + g(0) \quad (2.7)$$

Party 1 computes and shares $h_1 = (f + g)(1) = f(1) + g(1)$ while party 2 computes and shares $h_2 = (f + g)(2) = f(2) + g(2)$. The value of $(f + g)(0)$ can then be interpolated by fitting a polynomial through $(1, h_1)$ and $(2, h_2)$.

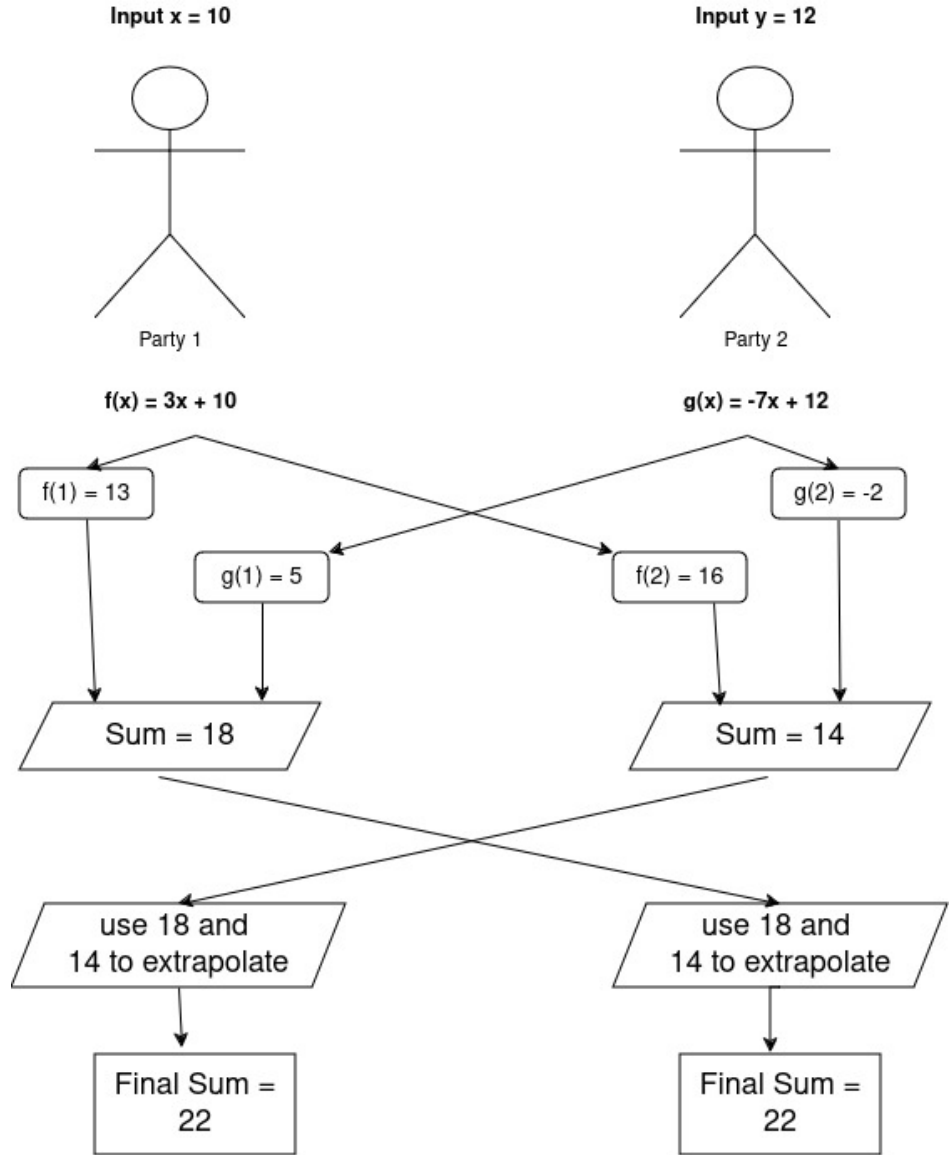


Figure 2.1: Example of secure addition using Shamir's secret sharing

2.5.5 Secure Point Inclusion Problem

Point inclusion is a classical problem in computational geometry. It can be defined as

Definition 2.5.2 *Classical Point Inclusion Problem*

Given a polygon and a point, determine if the point lies inside the polygon?

Some circumstances demand that the polygon and point come from different parties. However, both parties want to keep their input private. This problem is

a special case of multiparty secure computation where the inputs are point and polygon. The output is the relation between the point and the polygon.

Troncoso-Pastoriza et al. [28] describe many applications of secure point inclusion problem in the field of bio-metric authentication, positioning, database queries and watermarking.

we shall now define the secure version of point inclusion problem.

Definition 2.5.3 *Secure Point Inclusion Problem*

Two parties A and B have a point x and polygon P respectively. Without revealing their inputs to the other party, determine if the point lies inside the polygon securely. This can be seen as secure evaluation of a function f defined by:

$$f(x, P) = \begin{cases} 0 & \text{if } x \text{ lies outside of } P \\ 1 & \text{if } x \text{ lies inside of } P \end{cases}$$

Atallah and Du [9] use the following linear time algorithm for point inclusion problem which they modify to make secure using the several MPC primitives. We first describe this algorithm and then use the idea of secure evaluation using Shamir's secret sharing to implement the secure point inclusion problem in python.

We firstly present the non secure version in which we can assume that both the polygon and point are with the same individual.

Algorithm 1 Point Inclusion Algorithm [9]

Require: polygon \mathbb{P} and point (α, β)

$l \leftarrow$ leftmost vertex of Polygon

$r \leftarrow$ rightmost vertex of Polygon

$f_i(x, y) \quad i = 1, 2, \dots, m \leftarrow$ equation of lower line boundary of \mathbb{P} .

$f_i(x, y) \quad i = m + 1, m + 2, \dots, n \leftarrow$ equation of upper line boundary of \mathbb{P} .

for $i = 1, i \leq n, i = i + 1$ **do**

if $i \leq m$ and $f_i(\alpha, \beta) \leq 0$ **then**

\triangleright return False if point below lower boundary

return False

end if

if $i > m$ and $f_i(\alpha, \beta) \geq 0$ **then**

\triangleright return False if point is above upper boundary

return False

end if

\triangleright returns True if point lies inside polygon

return True

end for

Worked out Example

Consider the Polygon $\mathbb{P} = ABCDE$ defined by the points

- $A = (-3, 2)$
- $B = (3, 3)$
- $C = (5, 1)$
- $D = (-2, -1)$
- $E = (1, -2)$

We want to see whether the points $F = (5, -3)$ and $G(2, 1)$ lie in the polygon \mathbb{P} . We can visualize this using the below graph.

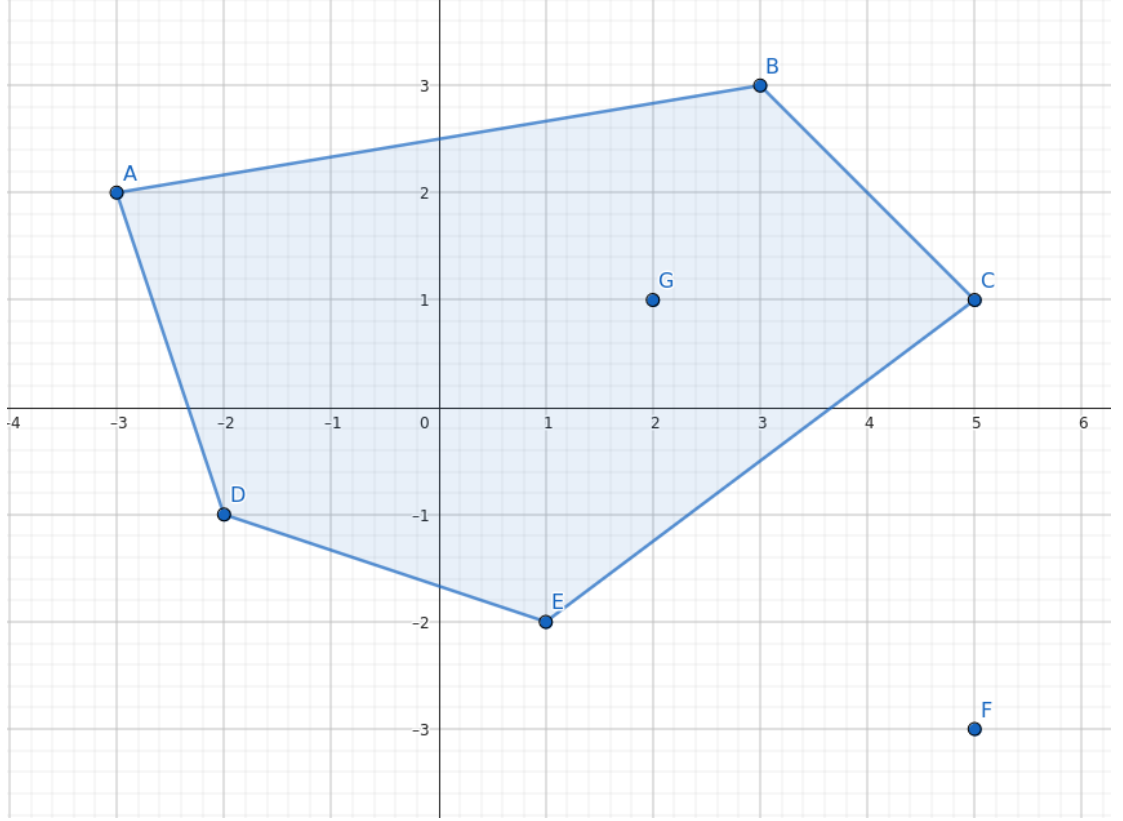


Figure 2.2: Graph with Polygon and Points
(generated with GeoGebra®)

We can compare and see that the point with smallest and largest abscissa (x-coordinate) are A and C respectively. Thus the A is the leftmost while C is the rightmost point which can be easily seen from the graph.

Now We have three lines (i.e, AD , DE and EC) that define the lower boundary and two lines (i.e, AB and BC) that define the upper boundary. We now need to determine the equations $f_i(x, y)$ and compute $f(\alpha, \beta)$.

Let $f(x, y) = 0$ be the equation of the line defined using points (x_1, y_1) and (x_2, y_2) . Using the two point formula for a line we get

$$f(x, y) = (y - y_1) - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) \quad (2.8)$$

Thus we get that: Lower Boundary consists of:

$$\text{Line } AD : f_1(x, y) = y + 3x + 7 \quad (2.9)$$

$$\text{Line } DE : f_2(x, y) = \frac{3y + x + 5}{3} \quad (2.10)$$

$$\text{Line } EC : f_3(x, y) = \frac{4y - 3x + 11}{4} \quad (2.11)$$

and the upper boundary consists of:

$$\text{Line } AB : f_4(x, y) = \frac{6y - x - 11}{6} \quad (2.12)$$

$$\text{Line } BC : f_5(x, y) = y + x - 6 \quad (2.13)$$

Now to test for point $F(5, -1)$, we evaluate the above equations at point F and G to get

Boundary	Equation	Evaluated at			
		F(5,-3)		G(2,1)	
Lower	Line AD: f_1	19	>0	12	>0
Lower	Line DE: f_2	0.33	>0	3.33	>0
Lower	Line EC: f_3	-4	<0	2.25	>0
Upper	Line AB: f_4	-5.67	<0	-1.167	<0
Upper	Line BC: f_5	-4	<0	-3	<0

Table 2.1: Tabulated values for the workout example of line point inclusion

As expected for the point G which lies in the polygon, we can see that the values of f_i evaluated at G , we have positive values for lower boundary and positive values for all upper boundary points.

However point F is less than zero for line EC which lies in lower boundary. We can clearly see this in the graph 2.2.

In the next chapter, we shall utilize this base algorithm combined with Shamir's secret sharing scheme to implement solution of the secure point inclusion problem in python using MPyC.

Chapter 3

Computer Implementation

3.1 Description of MPyC

A short introduction to MPyC has been presented in section 1.3.7. In this section, we shall be taking a look at the basics of how MPyC works.

To facilitate secure computation, MPyC supports secure types which are used to store inputs or any intermediate values that no single party should have access to. The secure type are stored in secret shared form, where each party has access to exactly one share of the value as per 'Shamir's Secret Sharing Scheme' as described in section 2.5.3.

Hence to supports n -party computation with up to t passive corruptions, we must utilize a $(t + 1, n)$ threshold scheme.

However MPyC places the restriction that [16]

$$0 \leq t \leq \frac{m - 1}{2} \quad (3.1)$$

We note that when $t = 0$, n -party computation is simply m parties computing with shared inputs but with no expectation of privacy.

3.2 Outline of Program

The program has the following parties involved.

Party	Role	Input
1	Alice	Point
2	Bob	Polygon
others	Trusted Helper	None

Table 3.1: Roles and Inputs in Secure Point Inclusion

The program needs at least three parties to run. Note that this program can be run with more than 3 parties. In fact, that makes it easier to maintain honest majority.

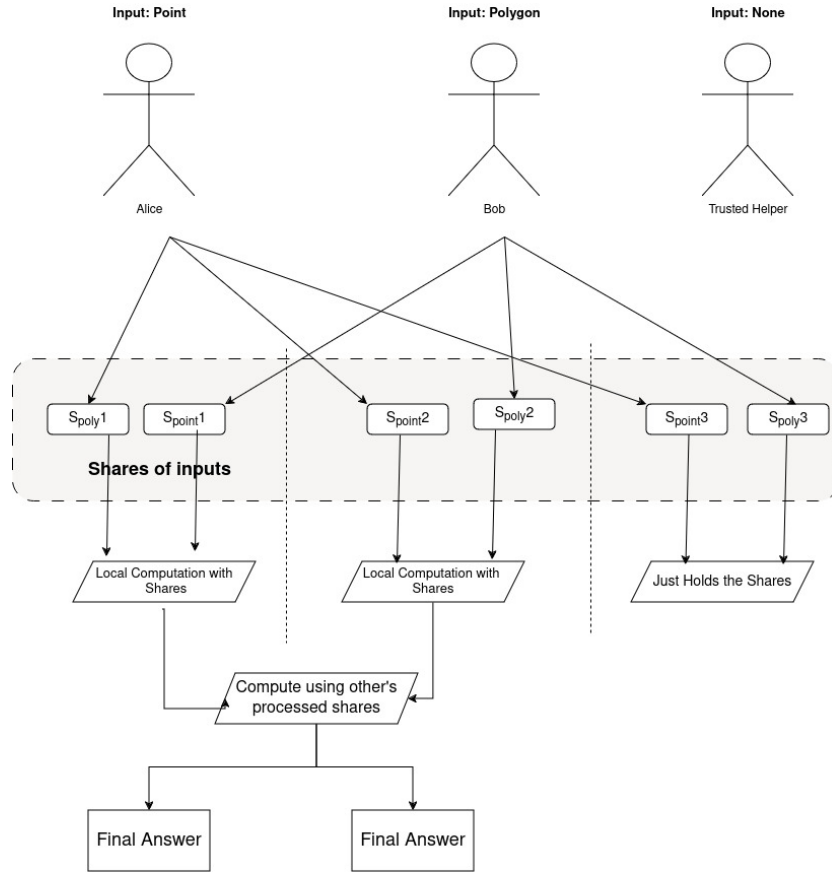


Figure 3.1: Flowchart of Implementation

3.3 Demonstration

3.3.1 Input : Point

The point to be tested is stored in a file named 'point.csv'. It contains a tuple containing x and y coordinate of the point.

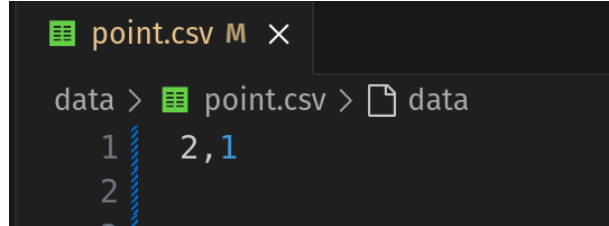


Figure 3.2: Example point.csv File

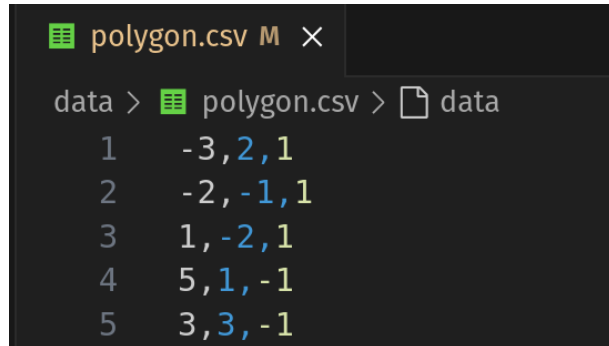
3.3.2 Input : Polygon

The polygon to be input into the algorithm is also stored as a file named 'polygon.csv'. The file should consists of triples such that the first two values represent vertices of the polygons while the third value of each tuple represents if the edges belong to lower boundary or the upper boundary.

The polygon does need to be preprocessed by the party with the polygon to conform to the following convention:

- The first entry must be the leftmost point.
- The points must be in cyclic order.
- The first m points must belong to lower boundary.

The edges when considered in cyclic order of vertex in the file need to be labelled for the algorithm to determine which party of the boundary they lie with. The convention to label edge with 1 if it belongs to lower boundary and -1 if it belongs to the upper boundary.



```
data > polygon.csv > data
1 -3, 2, 1
2 -2, -1, 1
3 1, -2, 1
4 5, 1, -1
5 3, 3, -1
```

Figure 3.3: Example polygon.csv File

3.3.3 Running the code

Although the protocol requires only two parties, its implementation using MPyC which requires us to have at least one trusted helper in order to ensure that the threshold is at least 2 and the privacy of the inputs can be achieved as long as everyone is honest.

We can run the program described in appendix A either locally or over the internet.

Running the Program locally

Enter the following commands in different terminals.

- In terminal 0 (trusted user)

```
python3 millionaire-problem.py -M2 -I0
```

- In terminal 1 (Alice or party with point)

```
python3 millionaire-problem.py -M2 -I0
```

- In terminal 2 (Bob or party with polygon)

```
python3 millionaire-problem.py -M2 -I0
```

Running the program over the internet

Make sure the various computers are accessible to each other and the relevant ports are open.

- In terminal 0 (trusted user)

```
python3 millionaire-problem.py -P localhost -P <IP:PORT of device 1> -P  
<IP:PORT of device 2> -I0
```

- In terminal 1 (Alice or party with point)

```
python3 millionaire-problem.py -P <IP:PORT of device 0> -P localhost -P
<IP:PORT of device 2> -I1
```

- In terminal 2 (Bob or party with polygon)

```
python3 millionaire-problem.py -P <IP:PORT of device 0> -P
<IP:PORT of device 1> -P localhost -I2
```

The program will execute and read the data files from the relevant parties and share the input shares. The computations are performed and then the output is provided to the parties.

```
point-inclusion-mpyc on 主main [!] via v3.10.12 (mpycvenv)
> python3 point-inclusion.py -M3 -I0
2024-07-02 12:46:49,764 Start MPyC runtime v0.10.1
2024-07-02 12:46:49,767 All 3 parties connected.
You are the trusted helper!
The solution is
None
2024-07-02 12:46:49,883 Stop MPyC -- elapsed time: 0:00:00.116|bytes sent: 53072
```

Figure 3.4: Execution for Trusted Helper

Note that the trusted helper provides no input and it also learns nothing about the output.

```
point-inclusion-mpyc on 主main [!] via v3.10.12 (mpycvenv) took 6s
> python3 point-inclusion.py -M3 -I1
2024-07-02 12:59:08,099 Start MPyC runtime v0.10.1
2024-07-02 12:59:10,453 All 3 parties connected.
You have a point as input! Your point is:
('2', '1')
The solution is
1
Note: 1 means Point lies inside Polygon!
Note: 0 Point lies outside Polygon!
2024-07-02 12:59:10,585 Stop MPyC -- elapsed time: 0:00:00.131|bytes sent: 53152
```

Figure 3.5: Execution by party containing point

```

point-inclusion-mpyc on main [!] via v3.10.12 (mpycenv) took 8s
> python3 point-inclusion.py -M3 -I2
2024-07-02 12:59:10,398 Start MPyC runtime v0.10.1
2024-07-02 12:59:10,453 All 3 parties connected.
You have a polygon as input! Your polygon is
('-3', '2')
('-2', '-1')
('1', '-2')
('5', '1')
('3', '3')
The solution is
1
Note: 1 means Point lies inside Polygon!
Note: 0 Point lies outside Polygon!
2024-07-02 12:59:10,585 Stop MPyC -- elapsed time: 0:00:00.131|bytes sent: 53566

```

Figure 3.6: Execution by party containing polygon

3.4 Limitations

- **Limited to two dimensions**

The current implementation is limited to two dimensional point and polygons. Point inclusion problem with higher degree polytopes are not supported in the current implementation.

- **Need of trusted helper**

The current implementation is done on MPyC which is based on Shamir's secret sharing scheme. Thus, we need at least one trusted helper to facilitate the secure protocol.

- **Requires Pre-ordered Vertices**

One limitation of the implementation is that it requires the party with the polygon to preprocess the vertices to affirm to the structure described in 3.3.2.

Chapter 4

Future Work and Recommendation

4.1 Future Work

There are many possible aspects for expansion and continuation of this work. Some of them are:

- Other algorithms in computational geometry may be considered for analysis and implementation.
- The problems may be extended to higher dimensions.
- The analysis of communication complexity of the protocols may be analyzed.
- Implementation in other frameworks which don't rely on Shamir's secret sharing scheme may be undertaken eliminating the need of trusted helper.

4.2 Conclusion

Multiparty secure computation has made it possible to maintain the privacy of the inputs while at the same time computing some function using those inputs. The idea of Shamir's secret sharing makes it possible to devise such a scheme. We have used MPyC to implement secure point inclusion. Such implementations can serve as a great way to experiment with these ideas and prototype new secure evaluation protocols.

Appendix A

Python Implementation of Secure Point Inclusion Problem in MPyC

```
""" Two party Point Inclusion Problem in 2D

Alice has has a point  $z = (a,b)$ .
Bob has a polygon.
We need a trusted helper.

Author: Mukesh Tiwari
Date: 27 Jun 2024
"""

#imports
from mpyc.runtime import mpc
import csv

#define a function that securely evaluates point line reln
def sec_point_line(x1,y1,x2,y2,x3,y3):
    # (x1,y1) and (x2,y2) define a line
    # (x3,y3) is the point
    val1 = (y3-y1)*(x2-x1) - (y2-y1)*(x3-x1)
    val2 = (x2-x1)

    tmp1 = mpc.if_else(val1>0,1,0)
    tmp2 = mpc.if_else(val2>0,1,0)
```

```

tmp3 = mpc.if_else(val1 == 0,1,0)
tmp4 = mpc.if_else(val2 == 0,1,0)
tmp5 = mpc.if_else(val1<0,1,0)
tmp6 = mpc.if_else(val2<0,1,0)

case1 = (tmp1 * tmp2) + (tmp5*tmp6)
case2 = (tmp1 * tmp6) + (tmp2*tmp5)

res1 = mpc.if_else(case1 > 0,1,-1)
res2 = mpc.if_else(case2 > 0,-1,1)

#This actually does return the correct value!!!
# val = val1/val2
# if val > 0 -> 1
# if val = 0 -> 0
# if val < 0 -> -1
return (res1 + res2)/2

async def main():

    secint = mpc.SecInt()

    await mpc.start()

    #define the variables and initialize them to None

    #point is a 2-tuple
    # point[0] = x-coordinate
    # point[1] = y -ordinate
    point = [None,None]
    len_polygon = None
    tmp = None

    #define role for the parties
    role = mpc.pid

    #if pid is neither 1 or 2, then you are trusted third party.
    if not (role in [1,2]):
        print('You are the trusted helper!')

```

```

# if pid is one, then you are Alice and thus have a point
if role == 1:
    print("You have a point as input! Your point is: ")
    #open the csv file containing the point's coordinates
    with open('data/point.csv') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            point[0] = int(row[0])
            point[1] = int(row[1])
            print((row[0],row[1]))

#if pid is two, you are Bob and have a polygon.
# the polygon is assumed to have the following properties:
# - the first entry is leftmost point.
# - the points are in cyclic order
# - the first m points belong to lower boundary
if role == 2:
    print("You have a polygon as input! Your polygon is ")
    #open the csv file containing the polygon's coordinates
    with open('data/polygon.csv') as csvfile:
        reader = csv.reader(csvfile)
        #count the number of points in polygon
        tmp = 0
        for row in reader:
            tmp = tmp + 1
            print((row[0],row[1]))

# Bob (Party 2) sends the number to vertices to all parties.
len_polygon = await mpc.transfer(tmp,2)

#initialize the polygon to None:
polygon = [[None,None]]*len_polygon
point_line_reln = [None]*len_polygon

#for now lets assume we have the polygon boundary
# 1 means lower; -1 means upper
polygon_boundary = [None]*len_polygon

#define the polygon using file:
#if pid is two, you are Bob and have a polygon.

```



```

if role == 2:
    #initialize polygon
    polygon = []
    #open the csv file containing the polygon's coordinates
    with open('data/polygon.csv') as csvfile:
        reader = csv.reader(csvfile)
        i = 0
        for row in reader:
            polygon.append([int(row[0]),int(row[1])])
            polygon_boundary[i] = int(row[2])
            i = i + 1

#convert the polygon to securepolygon

# secpolygon = []
# for vertex in polygon:
#     secpolygon.append([secint(vertex[0]),secint(vertex[1])
#                        ])

#Bob (party 2) securely shares the secure polygon
x = [None]*len_polygon
for i in range(len_polygon):
    # x[i] is the ith vertex
    x[i] = mpc.input([secint(polygon[i][0]),secint(polygon[i]
                                                    [1])],2)

#Alice (party 1) securely shares the point
sec_point = [None,None]
sec_point[0] = mpc.input(secint(point[0]),1)
sec_point[1] = mpc.input(secint(point[1]),1)

#Bob (party 2) securely shares the polygon boundary
sec_polygon_boundary = [None]*len_polygon
for i in range(len_polygon):
    sec_polygon_boundary[i] = mpc.input(secint(
                                                polygon_boundary[i]),2)

# take each edge and determine reln with the line
for i in range(len_polygon):

```

```

# the next point
j = (i+1) % len_polygon

# see if any is type None and if so skip the computation
is_none = [isinstance(x, type(None)) for x in [x[i][0],x[
i][1],x[j][0],x[j][1],
sec_point[0],sec_point[1
]]]

if not any(is_none):
    point_line_reln[i] = sec_point_line(x[i][0],x[i][1],x[
j][0],x[j][1],
sec_point[0],
sec_point[1])

#determine if the point is inside or not
result = 0
for i in range(len_polygon):
    if not (point_line_reln[i] is None or
sec_polygon_boundary[i]
is None):
        result = result + point_line_reln[i] *
sec_polygon_boundary
[i]

verdict = mpc.if_else(result == len_polygon,1,0)

#print the result but only give output to party 1 and 2
print("The solution is ")
print(await mpc.output(verdict,[1,2]))

# print(await mpc.output(verdict)) for parties 1 and 2
if role in [1,2]:
    print("\nNote: 1 means Point lies inside Polygon!")
    print("Note: 0 Point lies outside Polygon!")

await mpc.shutdown()

mpc.run(main())

```

References

- [1] D. Evans, V. Kolesnikov, and M. Rosulek, “A Pragmatic Introduction to Secure Multi-Party Computation,”
- [2] D. Davies, “A brief history of cryptography,” *Information Security Technical Report*, vol. 2, pp. 14–17, Jan. 1997.
- [3] A. Patra, *Studies On Verifiable Secret Sharing, Byzantine Agreement And Multiparty Computation*. PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
- [4] A. Shamir, R. L. Rivest, and L. Adleman, “Mental Poker,” *The Mathematical Gardner*.
- [5] A. C. Yao, “Protocols for Secure Computations,”
- [6] O. Goldreich, S. Micali, and A. Wigderson, “How to play ANY mental game,” in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC ’87, (New York, NY, USA), pp. 218–229, Association for Computing Machinery, Jan. 1987.
- [7] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, “Secure Multiparty Computation Goes Live,” in *Financial Cryptography and Data Security* (R. Dingledine and P. Golle, eds.), vol. 5628, pp. 325–343, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.
- [8] A. Lapets, F. Jansen, K. D. Albab, R. Issa, L. Qin, M. Varia, and A. Bestavros, “Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities,” Jan. 2018.

- [9] M. J. Atallah and W. Du, “Secure Multi-party Computational Geometry,” in *Algorithms and Data Structures* (G. Goos, J. Hartmanis, J. Van Leeuwen, F. Dehne, J.-R. Sack, and R. Tamassia, eds.), vol. 2125, pp. 165–179, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.
- [10] LUO. Yong-long, HUANG. Liu-sheng, XU. Wei-jiang, and JING. Wei-wei, “A Protocol for Privacy-Preserving Intersect-Determination of Two Polygons,” *ACTA ELECTRONICA SINICA*, vol. 35, pp. 685–691, Apr. 2007.
- [11] T. Geng, S. Luo, Y. Xin, X. Du, and Y. Yang, “Research on Secure Multi-party Computational Geometry,” in *Information Computing and Applications* (B. Liu and C. Chai, eds.), (Berlin, Heidelberg), pp. 322–329, Springer, 2011.
- [12] S. Basu, H. A. Khorasgani, H. K. Maji, and H. H. Nguyen, “Geometry of Secure Two-party Computation,” in *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, (Denver, CO, USA), pp. 1035–1044, IEEE, Oct. 2022.
- [13] L. Shundong, W. Chunying, W. Daoshun, and D. Yiqi, “Secure multiparty computation of solid geometric problems and their applications,” *Information Sciences*, vol. 282, pp. 401–413, Oct. 2014.
- [14] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, “SoK: General Purpose Compilers for Secure Multi-Party Computation,” in *2019 IEEE Symposium on Security and Privacy (SP)*, (San Francisco, CA, USA), pp. 1220–1237, IEEE, May 2019.
- [15] M. Dahlia, N. Noam, P. Benny, and S. Yaron, “Fairplay — A Secure Two-Party Computation System,”
- [16] B. Schoenmakers, “MPyC – Python Package for Secure Multiparty Computation,”
- [17] M. Geisler, “Cryptographic Protocols: Theory and Implementation,”
- [18] S. Micali and P. Rogaway, “Secure Computation,” in *Advances in Cryptology — CRYPTO ’91* (J. Feigenbaum, ed.), (Berlin, Heidelberg), pp. 392–404, Springer, 1992.

- [19] Y. Lindell, “Secure multiparty computation,” *Communications of the ACM*, vol. 64, pp. 86–96, Jan. 2021.
- [20] Y. Lindell, “Secure Multiparty Computation (MPC),”
- [21] R. L. Rivest, “Unconditionally Secure Commitment and Oblivious Transfer Schemes Using Private Channels and a Trusted Initializer,”
- [22] R. Canetti and A. Herzberg, “Maintaining Security in the Presence of Transient Faults,” in *Advances in Cryptology — CRYPTO ’94* (Y. G. Desmedt, ed.), (Berlin, Heidelberg), pp. 425–438, Springer, 1994.
- [23] M. O. Rabin, “How to Exchange Secrets with Oblivious Transfer,”
- [24] S. Even, O. Goldreich, and A. Lempel, “A randomized protocol for signing contracts,” *Communications of the ACM*, vol. 28, pp. 637–647, June 1985.
- [25] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, pp. 612–613, Nov. 1979.
- [26] S. Rao and D. Tse, “Polynomials : Discrete Mathematics and Probability Theory.”
- [27] J. L. Lagrange, *Lectures on Elementary Mathematics*. 1736/1813.
- [28] J. R. Troncoso-Pastoriza, S. Katzenbeisser, M. Celik, and A. Lemma, “A secure multidimensional point inclusion protocol,” in *Proceedings of the 9th Workshop on Multimedia & Security, MM&Sec ’07*, (New York, NY, USA), pp. 109–120, Association for Computing Machinery, Sept. 2007.